Introduction to Graph Algorithms and Graph Representation

Week-09, Lecture-01

Course Code: CSE221
Course Title: Algorithms
Program: B.Sc. in CSE

Course Teacher: Tanzina Afroz Rimi
Designation: Lecturer
Email: tanzinaafroz.cse@diu.edu.bd

Graphs

A graph is a mathematical and visual representation of a set of objects, often referred to as "nodes" or "vertices," along with connections between pairs of those objects, known as "edges." Graphs are widely used in various fields, including mathematics, computer science, social sciences, and biology, to model relationships, networks, and structures. Graphs provide a flexible and intuitive way to represent and analyze complex systems and interactions.

There are two primary components of a graph:

1. **Nodes (Vertices)**: Nodes are the individual entities or elements in a graph. Each node can represent a person, a place, an object, a concept, or any other entity of interest.

2. **Edges**: Edges are the connections or relationships between nodes in a graph. An edge between two nodes indicates that there is some kind of interaction, association, or link between those nodes. Edges can be directed (have a specific direction) or undirected (no direction), weighted (have a numerical value or weight associated with them) or unweighted.

Graph

Graphs can take various forms based on their characteristics and the type of relationships they represent:

1. **Undirected Graph**: In an undirected graph, edges have no direction. If there's an edge between nodes A and B, it means that the relationship is symmetric, and the connection applies to both nodes equally.

2. **Directed Graph (Digraph)**: In a directed graph, edges have a specific direction from one node (the source) to another (the target). This allows for representing asymmetric relationships where one node influences or directs another.

3. **Weighted Graph**: In a weighted graph, edges have associated weights or numerical values. These weights can represent strengths, distances, costs, or any other relevant attribute of the relationship.

4. **Unweighted Graph**: In an unweighted graph, edges do not have associated weights; they simply represent the presence of a relationship.

5. **Cyclic Graph**: A graph that contains at least one cycle, which is a sequence of nodes and edges that starts and ends at the same node.

6. **Acyclic Graph**: A graph without any cycles.

Graph

Trivial Graph: A graph with one vertex and no edges.

Complete Graph: Every pair of distinct vertices is connected by a unique edge.

Bipartite Graph: Vertices can be divided into two disjoint sets such that every edge connects a vertex in one set to a vertex in the other set.

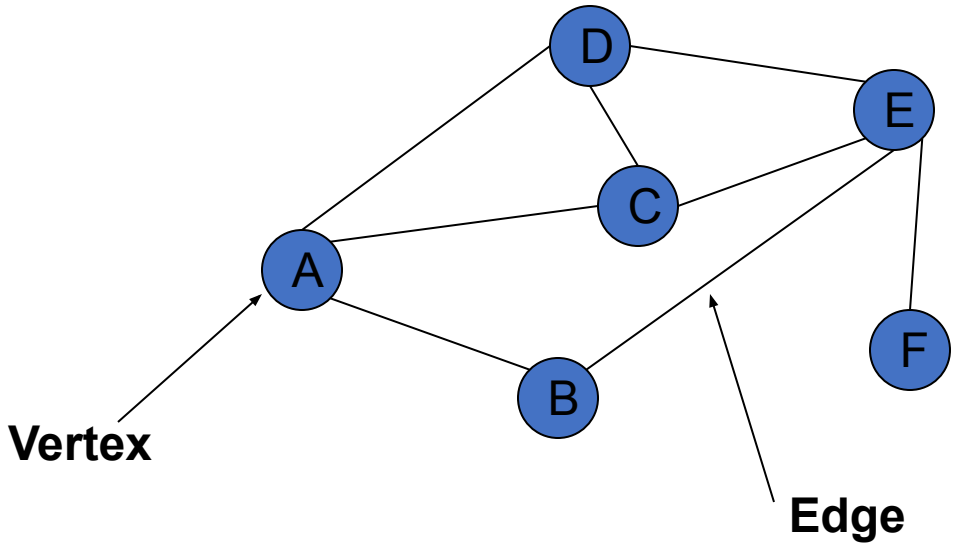
Connected Graph: There is a path between every pair of vertices.

Disconnected Graph: Not all vertices are connected by paths.

Regular Graph: Every vertex has the same number of connecting edges.

Graphs

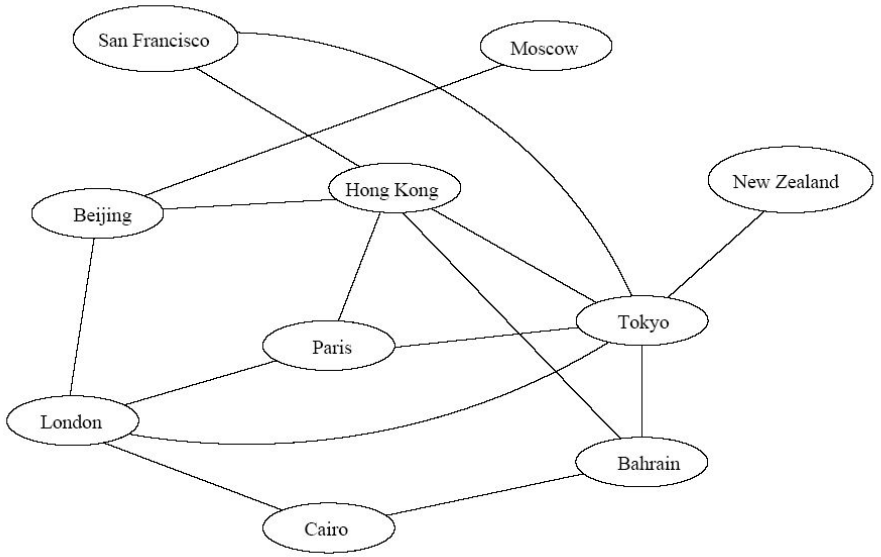
- Extremely useful tool in modeling problems
- Consist of:
  - Vertices
  - Edges



Vertices can be considered “sites” or locations.

Edges represent connections.

Application

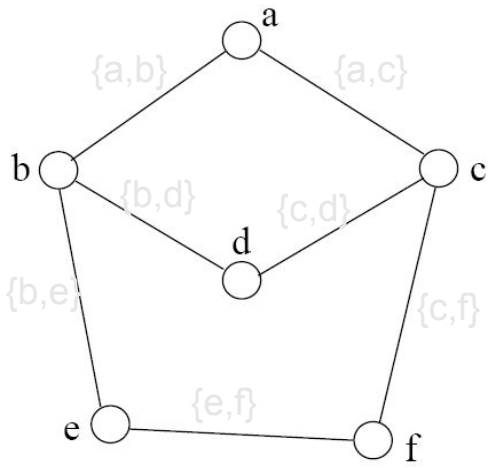


Air flight system

- Each vertex represents a city
- Each edge represents a direct flight between two cities
- A query on direct flights = a query on whether an edge exists
- A query on how to get to a location = does a path exist from A to B
- We can even associate costs to edges (weighted graphs), then ask "what is the cheapest path from A to B"

Definition

- A graph G=(V, E) consists a set of vertices, V, and a set of edges, E.
- Each edge is a pair of *(v, w)*, where v, w belongs to V
- If the pair is unordered, the graph is undirected; otherwise it is directed



$$V = \{a, b, c, d, e, f\}$$

$$E = \{\{a, b\}, \{a, c\}, \{b, d\}, \{c, d\}, \{b, e\}, \{c, f\}, \{e, f\}\}$$

An undirected graph

Definition

- Complete Graph
  - How many edges are there in an N-vertex complete graph?
- Bipartite Graph
  - What is its property? How can we detect it?
- Path
- Tour
- Degree of a vertices
  - Indegree
  - Outdegree
  - Indegree+outdegree = Even (why??)

Graph Variations

- Variations:
  - A *connected graph* has a path from every vertex to every other
  - In an *undirected graph:*
    - Edge (u,v) = Edge (v,u)
    - No self-loops
  - In a *directed* graph:
    - Edge (u,v) goes from vertex u to vertex v, notated u→v

Graph Variations

- More variations:
  - A *weighted graph* associates weights with either the edges or the vertices
    - E.g., a road map: edges might be weighted w distance
  - A *multi-graph* allows multiple edges between the same vertices
    - E.g., the call graph in a program (a function can get called from multiple points in another function)

Graphs

- We will typically express running times in terms of |E| and |V| (often dropping the |'s)
  - If $|E| \approx |V|^2$ the graph is *dense*
  - If $|E| \approx |V|$ the graph is *sparse*
- If you know you are dealing with dense or sparse graphs, different data structures may make sense

Graph Representation

- Two popular computer representations of a graph. Both represent the vertex set and the edge set, but in different ways.

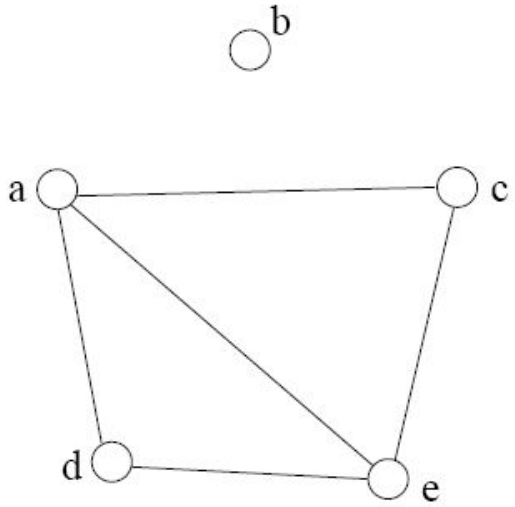
1. Adjacency Matrix

   Use a 2D matrix to represent the graph

2. Adjacency List

   Use a 1D array of linked lists

Adjacency Matrix



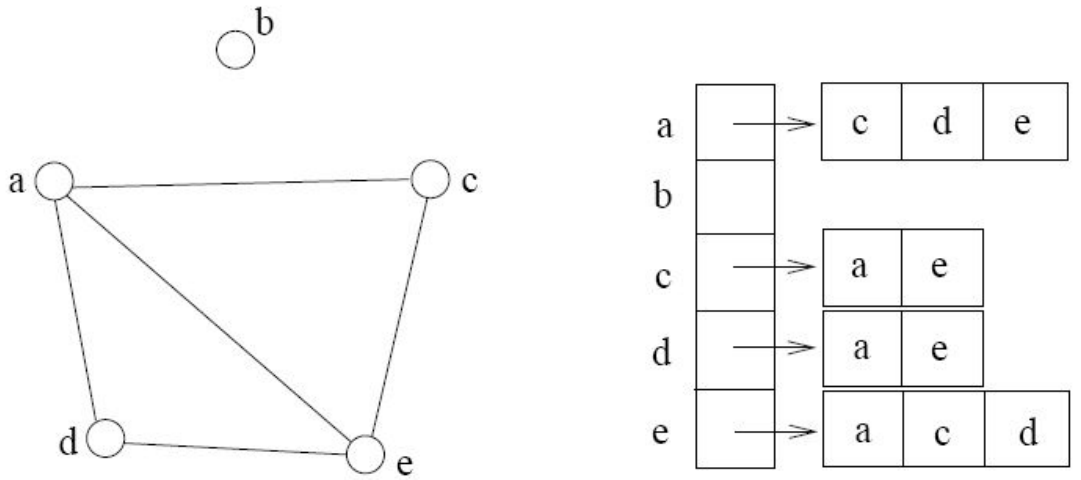
	a	b	c	d	e
a	0	0	1	1	1
b	0	0	0	0	0
c	1	0	0	0	1
d	1	0	0	0	1
e	1	0	1	1	0

- 2D array A[0..n-1, 0..n-1], where ***n*** is the number of vertices in the graph
- Each row and column is indexed by the vertex id
  - e,g a=0, b=1, c=2, d=3, e=4
- *A[i][j]=1* if there is an edge connecting vertices *i* and *j*; otherwise, *A[i][j]=0*
- The storage requirement is $\Theta(n^2)$. It is not efficient if the graph has few edges. An adjacency matrix is an appropriate representation if the graph is dense: $|E|=\Theta(|V|^2)$
- We can detect in O(1) time whether two vertices are connected.

Simple Questions on Adjacency Matrix

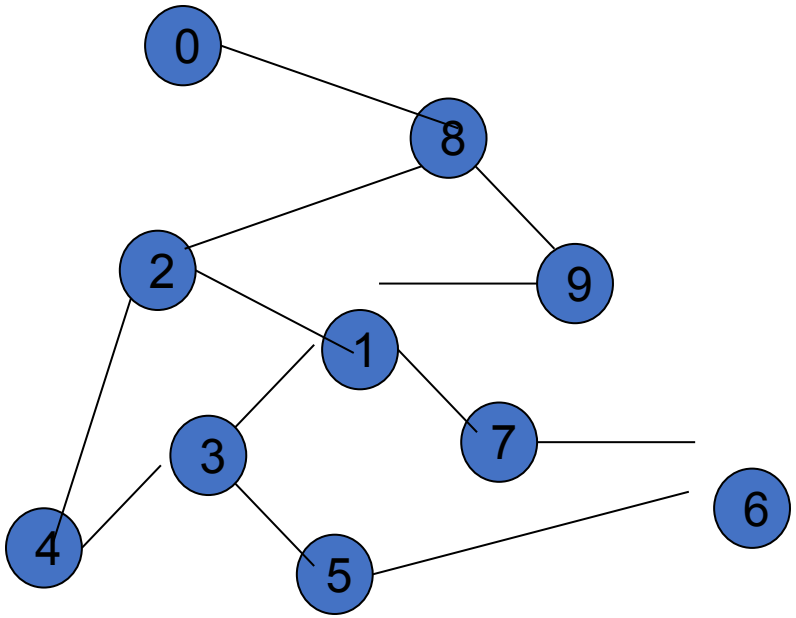
- Is there a direct link between A and B?
- What is the indegree and outdegree for a vertex A?
- How many nodes are directly connected to vertex A?
- Is it an undirected graph or directed graph?
- Suppose ADJ is an NxN matrix. What will be the result if we create another matrix ADJ2 where ADJ2=ADJxADJ?

Adjacency List



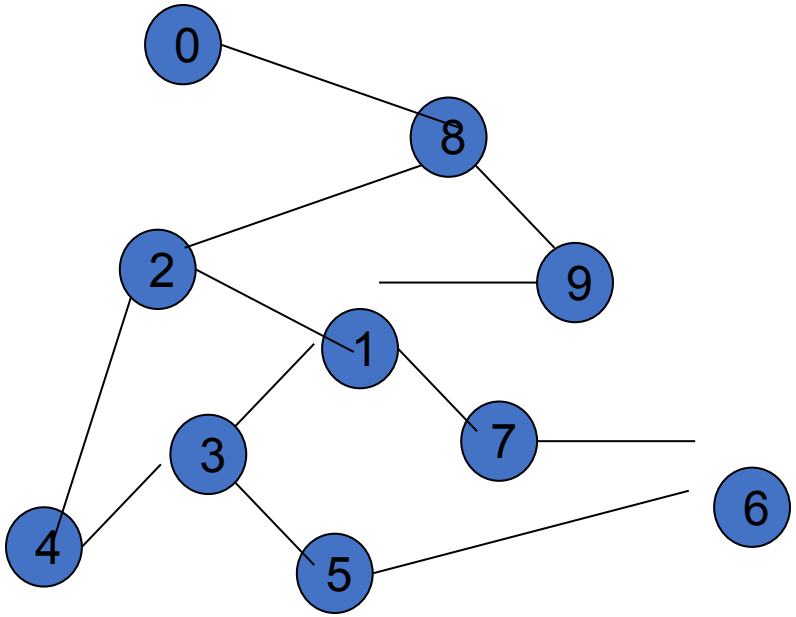
- If the graph is not dense, in other words, sparse, a better solution is an adjacency list
- The adjacency list is an array A[0..n-1] of lists, where n is the number of vertices in the graph.
- Each array entry is indexed by the vertex id
- Each list *A[i]* stores the ids of the vertices adjacent to vertex *i*

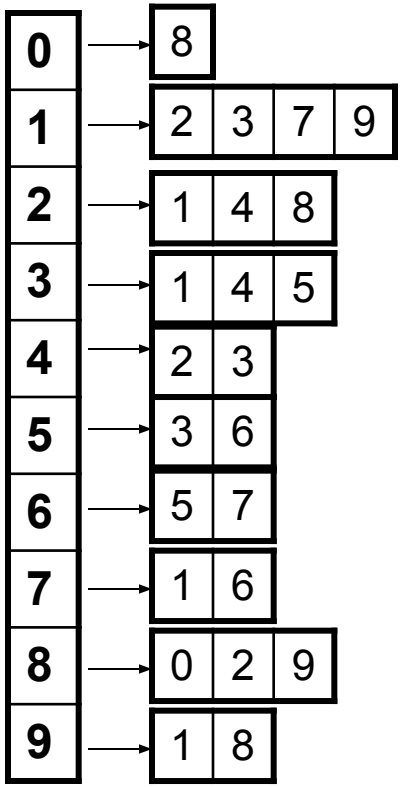
Adjacency Matrix Example



	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0	1	0	1
2	0	1	0	0	1	0	0	0	1	0
3	0	1	0	0	1	1	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0
5	0	0	0	1	0	0	1	0	0	0
6	0	0	0	0	0	1	0	1	0	0
7	0	1	0	0	0	0	1	0	0	0
8	1	0	1	0	0	0	0	0	0	1
9	0	1	0	0	0	0	0	0	1	0

Adjacency List Example





Storage of Adjacency List

- The array takes up $\Theta(n)$ space
- Define degree of v, deg(v), to be the number of edges incident to v. Then, the total space to store the graph is proportional to:

$$\sum_{\text{vertex } v} \deg(v)$$

- An edge $e=\{u,v\}$ of the graph contributes a count of 1 to deg(u) and contributes a count 1 to deg(v)
- Therefore, $\Sigma_{\text{vertex } v} \deg(v) = 2m$, where m is the total number of edges
- In all, the adjacency list takes up $\Theta(n+m)$ space
  - If $m = O(n^2)$ (i.e. dense graphs), both adjacent matrix and adjacent lists use $\Theta(n^2)$ space.
  - If $m = O(n)$, adjacent list outperform adjacent matrix

- However, one cannot tell in O(1) time whether two vertices are connected

Adjacency List vs. Matrix

- **Adjacency List**
  - More compact than adjacency matrices if graph has few edges
  - Requires more time to find if an edge exists

- **Adjacency Matrix**
  - Always require n^2 space
    - This can waste a lot of space if the number of edges are sparse
  - Can quickly find if an edge exists

Path between Vertices

- A path is a sequence of vertices (v_0, v_1, v_2,… v_k) such that:
  - For $0 \leq i < k$, $\{v_i, v_{i+1}\}$ is an edge

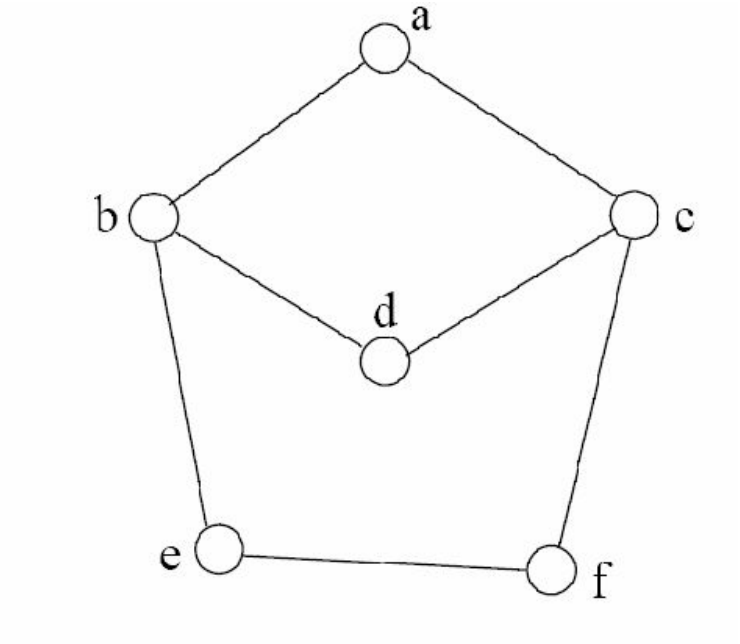
Note: a path is allowed to go through the same vertex or the same edge any number of times!

- The length of a path is the number of edges on the path

Types of paths

- A path is simple if and only if it does not contain a vertex more than once.
- A path is a cycle if and only if $v_0 = v_k$
  - The beginning and end are the same vertex!
- A path contains a cycle as its sub-path if some vertex appears twice or more

Path Examples



Are these paths?

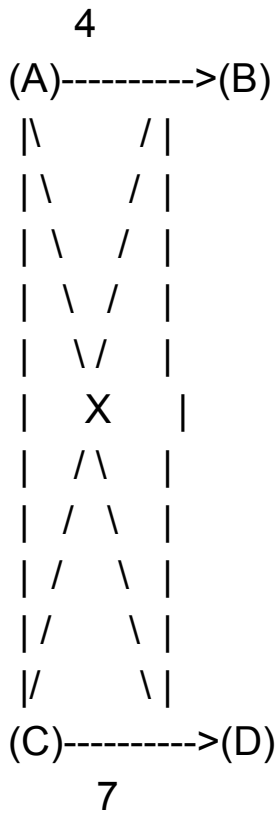
Any cycles?

What is the path's length?

1. {a,c,f,e}
2. {a,b,d,c,f,e}
3. {a, c, d, b, d, c, f, e}
4. {a,c,d,b,a}
5. {a,c,f,e,b,d,c,a}

Practice problem

Problem: Consider the following weighted directed graph:



Practice problem

1. List the nodes (vertices) in the graph.
2. List the edges in the graph along with their weights.
3. Calculate the out-degree and in-degree of each node.
4. Determine the shortest path from node A to node D using Dijkstra's algorithm.
5. Determine the shortest path from node A to node D using Bellman-Ford algorithm.
6. Identify any negative weight cycles in the graph.
7. Calculate the minimum spanning tree using Prim's algorithm.
8. Calculate the minimum spanning tree using Kruskal's algorithm.

Textbooks & Web References

- Text Book (Chapter 22)
- Reference book i (Chapter 4)
- www.geeksforgeeks.org
- www.codeforces.com

Thank you

&

Any question?